Trabalho Intermediário

Victor Marcius Magalhães Pinto

1 Descrição da Tarefa

teste

2 Implementação de funções intermediárias

Para a execução da tarefa, algumas funções auxiliares foram implementadas. A implementação das mesmas pode ser visto a seguir.

```
> checkAcc <- function(result, correct) {
+
+     if (length(result) != length(correct)){
+         cat("[checkAcc Error] Sizes of result and correct are diferent! Please
+         return(NULL)
+     }
+
+     count <- 0
+     for (i in seq(length(result))) {
+         if (result[i] == correct[i]) count <- count + 1
+     }
+
+     percCorrect <- count / length(result) * 100
+
+     return(c(count, percCorrect))
+ }
> MostraImagem <- function( x )
+ {
+
+     img <- matrix( x, nrow=64 )
+     cor <- rev( gray(50:1/50) )
+     image( rotate( img ), col=cor )
+ }
> # MostraImagem( faces[1,] )
>
> pdfnvar <- function(x,m,K,n) {
+
```

```
+     y <- ((1/(((2*pi)^n*(det(K)))))*exp(-0.5*(t(x-m)%*%(pseudoinverse(K))%*%(x-
+     return(y)
+
+ }
> bayes <- function(xtreino, ytreino, nSamples, xteste, yteste) {
+
+     Mcov <- list()
+     nClass <- nrow(xtreino) / nSamples
+     classe <- 1
+     meansList <- list()
+     nFeatures <- ncol(xtreino)
+
+     for (i in seq(1,nrow(xtreino), nSamples)) {
+         Mcov[[classe]] <- cov(xtreino[i:(i+nSamples-1),])
+         meansList[[classe]] <- colMeans(xtreino[i:(nSamples-1),])
+         classe <- classe + 1
+     }
+
+     pdf <- matrix(nrow = nrow(xteste), ncol = nClass)
+
+     for (i in seq(nrow(xteste))) {
+         for (j in seq(nClass)) {
+             pdf[i,j] <- pdfnvar(xteste[i,],meansList[[j]],Mcov[[j]],nFeatures)
+         }
+     }
+
+     predResult <- matrix(nrow = nrow(pdf))
+     predResult <- apply(pdf, 1,which.max)
+
+     results <- list()
+     checkAcc <- predResult - yteste
+     win <- sum(checkAcc == 0)
+     predWin <- win / length(yteste) * 100
+
+     results["predResult"]] <- predResult
+     results["predWin"]] <- predWin
+
+
+     return(results)
+ }
```

```
>
```

3 Execução do código

Primeiramente, foram carragadas as imagens contidas no pacote de faces. O pacote contém 40 classes de imagens, com 10 amostras cada.

3.1 Carregando os dados

```
> data( faces )
> faces <- t( faces )
> rotate <- function(x) t( apply(x, 2, rev) )
> y <- NULL
> for(i in 1:nrow(faces) )
+ {
+     y <- c( y, ((i-1) %/% 10) + 1 )
+ }
> # Nomeando os atributos
> nomeColunas <- NULL
> for(i in 1:ncol(faces) )
+ {
+     nomeColunas <- c(nomeColunas, paste("a", as.character(i), sep=".") )
+ }
> nomeLinhas <- NULL
> for(i in 1:nrow(faces)) {
+     nomeLinhas <- c(nomeLinhas, paste("face", as.character(y[i]),as.character(i
+ }
> colnames(faces) <- nomeColunas
> faces <- as.data.frame(faces, row.names = nomeLinhas)
> rm(nomeColunas)
> rm(nomeLinhas)
```

Cada imagem foi transformada, de uma matrix de 64x64 pixels, em um vetor de 4096 features, e inserida em um dataframe de 400 linhas.

3.2 Diminuindo o número de atributos com PCA

Em seguida, foi realizada a diminuição do número de atributos da base de dados utilizando o algoritmo do PCA. Em implementações anteriores, foi utilizada a função *preProcess* juntamente com a *predict* do pacote *caret*, o que nos informva o

número mínimo de atributos para uma exatidão de 95% de 123 features. Porém, o pacote apresentou problemas de funcionamento em dias posteriores, o que nos exigiu a utilização da função *prcomp*, que pode ser vista no trecho de código a serguir.

```
> facesPCAaux <- prcomp(faces, center=TRUE, retx=TRUE, scale=TRUE)
> facesPCA <- facesPCAaux$x[,1:200]
>
```

Observando a implementação da função do classificador de Bayes, e da geração de pdfs para amostras multivariáveis, uma matriz de correlação invertível é gerada apenas se o número de linhas for menor ou igual ao número de colunas, única possibilidade que não gera uma matriz singular quando da execução da função *solve*. Como são 10 amostras de cada classe, e para o trabalho será utilizada 5 amostras para treino e 5 para testes, o número de features máximo que pode ser utilizado, desta forma, para o correto funcionamento do algoritmo do classificador de Bayes, é de 4 features.

Porém, considerando matrizes não invertíveis, no momento do cálculo do pdf, podemos fazer uso de uma matriz pseudoinversa, através da função *pseudoinverse* do pacote *coropcor*. Isto nos permite utilizar mais features do que através da implementação anterior. Portanto, para as rotinas de trainamento, faremos uso de 200 features, metade do que oferecido como resultado da diminuição utilizando PCA.

3.3 Diminuindo o número de atributos com MDS

O mesmo procedimento, e a mesma limitação do número de features foi realizado para o me´todo MDS. O algoritmo pode ser visto a seguir.

```
> kMDS <- 200
> facesMDS <- cmdscale(dist(faces), k=kMDS)
>
```

3.4 Gerando sets de treino e teste

Como dito anteriormente, para o treinamento e classificação pelos algoritmos, foram utilizadas 5 amostras de cada face. A geração das amostras pdoe ser vista no trecho de código a seguir. A mesma possui uma alta complexidade de implementação, visto da utilizazão de loops internos, e de toda o algoritmo, é o trecho que consome mais tempo de execução

```
> dim_classe <- 10
> numClasses <- 400
> numAmostras <- 10
> seqN <- sample(numAmostras)
> porcAmostTrain <- 0.5
> N <- seqN[1:(porcAmostTrain*numAmostras)]
> nSamplesTrain <- length(N)
> n <- seqN[(porcAmostTrain*numAmostras+1):numAmostras]
> nSamplesTest <- length(n)
> xtreino <- c()
> xtreinoPCA <- c()
> xtreinoMDS <- c()
> ytreino <- c()
> xteste <- c()
> xtestePCA <- c()
> xtesteMDS <- c()
> yteste <- c()
> for(r in seq(1,numClasses,numAmostras)) {
+     for(i in N) {
+         xtreino <- rbind(xtreino, (faces[r+i-1,]))
+         xtreinoPCA <- rbind(xtreinoPCA, (facesPCA[r+i-1,]))
+         xtreinoMDS <- rbind(xtreinoMDS, (facesMDS[r+i-1,]))
+         ytreino <- c(ytreino,(y[r+i-1]))
+     }
+
+     for(i in n) {
+         xteste <- rbind(xteste, (faces[r+i-1,]))
+         xtestePCA <- rbind(xtestePCA, (facesPCA[r+i-1,]))
+         xtesteMDS <- rbind(xtesteMDS, (facesMDS[r+i-1,]))
+         yteste <- c(yteste,(y[r+i-1]))
+     }
+ }
>
```

3.5 Classificando com KNN e PCA

Começando então à realizar as classificações, foi feito uso da função *knn* para a classificação das amostras de teste, com base nos resultados oferecidos pelas amostras de treinamento. Como a função nos permite escolher o número de vizinhos a ser considerados para o correto estabelecimento, foram relizadas 10 rotinas de treinamento, variando esse parâmetro entre 1 e 10. A rotina utilizada pode ser

vista a seguir.

```
> resultKNNPCA <- c()
> for (i in seq(10)) {
+     separation <- knn(xtreinoPCA,xtestePCA,ytreino,k=i)
+     resultKNNPCA <- rbind(resultKNNPCA,matrix(c(i,(checkAcc(separation, yteste)
+ }
> colnames(resultKNNPCA) <- c("N","Accuracy")
> meanResultKNNPCA <- mean(resultKNNPCA[,2])
> results[["KNNPCA"]] <- resultKNNPCA
>
```

Os resultados da execução deste algoritmo podem ser vistos a seguir.

```
      N Accuracy
 [1,]  1     92.5
 [2,]  2     82.0
 [3,]  3     79.5
 [4,]  4     77.0
 [5,]  5     74.5
 [6,]  6     68.5
 [7,]  7     68.5
 [8,]  8     62.5
 [9,]  9     60.0
[10,] 10     62.5
```

3.6 Classificando com KNN e MDS